Software Consolidation as an Efficient Energy and Cost Saving Solution for a SaaS/PaaS Cloud Model

Alain Tchana[1(✉)], Noel De Palma[2], Ibrahim Safieddine[2], Daniel Hagimont[1], Bruno Diot[3,4], and Nicolas Vuillerme[3,5]

[1] University of Toulouse, Toulouse, France
alain.tchana@enseeiht.fr
[2] University of Joseph Fourier, Grenoble, France
noelde.palma@imag.fr
[3] IDS, Montceau-les-mines, France
b.diot@ids-assistance.com, bruno.diot@agim.eu
[4] AGIM, Université Grenoble Alpes, Saint-Martin-d'Hères, France
[5] Institut Universitaire de France, Paris, France

Abstract. Virtual machines (VM) are used in cloud computing environments to isolate different software. Virtualization enables live migration, and thus dynamic VM consolidation. This possibility can be used to reduce power consumption in the cloud. However, consolidation in cloud environments is limited due to reliance on VMs, mainly due to their memory overhead. For instance, over a 4-month period in a real cloud located in Grenoble (France), we observed that 805 VMs used less than 12 % of the CPU (of the active physical machines). This paper presents a solution introducing dynamic software consolidation. Software consolidation makes it possible to dynamically collocate several software applications on the same VM to reduce the number of VMs used. This approach can be combined with VM consolidation which collocates multiple VMs on a reduced number of physical machines. Software consolidation can be used in a private cloud to reduce power consumption, or by a client of a public cloud to reduce the number of VMs used, thus reducing costs. The evaluation was performed using both the SPECjms2007 benchmark and an enterprise LAMP benchmark on both a VMware private cloud and Amazon EC2 public cloud. The results show that our approach can reduce the energy consumed in our private cloud by about 40 % and the charge for VMs on Amazon EC2 by about 40.5 %.

Keywords: Cloud · Consolidation · Energy saving · Virtualization

1 Introduction

In recent years, cloud computing has emerged as one of the best solutions to host applications for companies or individual users. For these cloud customers (hereafter called clients), its pay-per-use model reduces the cost compared to

using internal IT resources. For cloud providers (hereafter called providers) one of the main challenges is limiting power consumption in their data centers. In 2010, for example, data centers consumed approximately 1.1–1.5 % of the world's energy [14]. Power consumption can be minimized by limiting the number of active physical machines (PM) through sharing the same PM between several software applications and providing dynamic software consolidation (filling unused resources by grouping software). This helps to balance the variable workload due to the departure of some software.

In this paper, we considered a SaaS/PaaS-based cloud model (e.g. RightScale [5]). This type of cloud provides a fully customizable environment, allowing clients, e.g. companies, to focus on applications. The cloud provider offers a software catalogue. Clients can select an application and request its start in a virtualized data center. The data center may belong either to the cloud provider (private cloud), or be part of a public cloud; alternatively it can be a mixture of the two (hybrid cloud). The cloud provider is responsible for managing the clients' software (scalability, highly-available, failover, etc.) while efficiently managing resources to reduce data center costs: power consumption when relying on its own private cloud, or resource charged for when using a public cloud.

Advances in virtualization make transparent dynamic consolidation possible in the cloud. Based on this technology, the cloud runs each software application on a separate virtual machine (VM). Many studies [12,16,23] have described algorithms providing software consolidation through the consolidation of VMs. However, this approach is not sufficient since an infinite number of VMs cannot be packed into a single PM, even when the VMs are underused and the PM has sufficient computation power. Indeed, as argued by [20], VM packing is limited by memory. **In this paper, we therefore propose a solution consolidating software onto VMs. This solution is complementary to VM consolidation**. Rather than dedicating one VM to each software, we propose that the same VM be shared between several software applications. This will fill the gaps remaining inside the VM, as mentioned earlier. **This strategy also reduces the total number of VMs. This is very important in a commercial cloud to reduce the charge for VMs**.

Software consolidation raises two main challenges that need to be addressed:

- Software isolation. Isolation ensures that if a software application fails it does not compromise the execution of another software application, it also stops software from "stealing" the resources allocated to another application.
- Software migration. Migration involves moving software from its current node to another node without interrupting the service offered by the software, and while avoiding Service Level Agreement (SLA) violations on the migrated software.

In this paper we focus only on the live software migration and consolidation. For software isolation, we rely on Docker [2]. We present a solution to consolidate software on VMs (Sect. 2) based on a Constraints Programming (CP) solver. **The genericity of the solution allows the integration of a range of**

live software migration mechanisms since this operation is specific to software. We evaluated our approach using the SPECjms2007 benchmark [6] and an enterprise Internet application benchmark (Sect. 4) in the context of a SaaS offering messaging and Internet software on a private VMware cloud and on the Amazon EC2 cloud. These evaluations showed that: (1) our approach results in reduced power consumption and costs; and (2) the efficient live migration algorithms implemented for JMS messaging and Internet web servers are viable. For the specific workload assessed, our solution reduces the power consumption in our private cloud by about 40 % when software consolidation is combined with VM consolidation. Running the same workload on Amazon EC2 leads to a reduction in VMs charged of about 40.5 %. The paper ends by presenting related work in Sect. 5 and a conclusion is provided in Sect. 6.

2 Software Consolidation

Like VMs, software consolidation is an NP-hard [13] problem. We presents a solution that allows software consolidation for SaaS/PaaS platform.

2.1 Solution Overview

We focus on software consolidation and migration. VM placement at start time is part of the consolidation problem. Figure 1 presents the key components of the system studied. Applications are isolated within VMs using Docker containers [2]. *QuotaComputer* determines the amount of resources required by each application. *MonitoringEngine* is responsible for gathering statistics for both VM and software from all *MonitoringAgent*s. The *ConsolidationManager* implements an online, reactive consolidation algorithm which acts as an infinite loop. It periodically:

1. Gets VMs and software status (quota consumption, which is an average of the most recent values) from the *MonitoringEngine*.
2. Checks if there are applications which need more resources and provides for them (relocation Algorithm 1).
3. Computes software assignment on VMs to minimize the number of VMs required to support all the software running. It also computes the reconfiguration plan (a set of software migrations) that must be performed for the ideal assignment to be achieved (Sect. 2.2).
4. And finally, runs (through the *LocalManager*) the reconfiguration plan.

At the end of each loop, VMs not running any software are terminated, either immediately in the case of a private cloud, or when its uptime is close to a multiple of Θ (the payment time unit) in a public cloud. In the latter case, a timer is started for each VM to be terminated so that it stops it before a new payment time unit starts. The timer is disabled when the VM is eligible to host a running application.

Before presenting our solution in detail, there follows a list of the notations used:

– $S = \{S_1, S_2, ..., S_m\}$ is the set of software types offered by the cloud.
– $\bar{S}_i = \{S_k, ...\}$ is the set of software, instances of which are collocate-able with an instance of S_i. This can be used to protect sensitive software from other potentially dangerous applications.
– For each VM vm_i, we consider three types of resources: cpu (vm_i^u), memory (vm_i^m) and IO bandwidth (vm_i^o).
– vm_i^γ is the cost of running the VM for a payment time unit Θ. This is considered when the SaaS/PaaS is placed on a public cloud.
– vm_i^{st} is the start time of vm_i.
– An instantiated VM is assigned an identifier (an integer). s_i^{vm} is the identifier of the VM running software s_i.
– len(vm_i) is the number of applications on vm_i.
– Like VMware does for VMs, we consider two levels of resource reservation for a software: the minimum quota and the maximum quota. $s_i^{u_{min}}$ and $s_i^{u_{max}}$ are, respectively, the minimum and the maximum cpu (or memory or IO) quota. The software starts with $s_i^{*_{min}}$ and increases stepwise until it reaches $s_i^{*_{max}}$. $s_i^{*_{cur}}$ denotes the current quota. Note that * is u, m or o.
– s_i^T is the acceptable service degradation threshold defined at start time for software s_i. It corresponds to its SLA.

The relocation algorithm described in Algorithm 1, checks if the current resources available for each application are insufficient, excessive or sufficient. If not, the software acquires more resources within its maximum quota. This operation can cause the software to be relocated to another VM (an existing one or a new one). On the other hand, if the software is wasting resources, its quota is reduced; the algorithm includes a clause to avoid the frequent transitions (yo-yo effect). The choice of the destination VM (on which software is to be relocated) does not need to be optimal. Indeed, the consolidation manager will correct the placement. This will be discussed in the next section, where the formalization of the software placement problem as a Constraint Satisfaction Problem (CSP) [9] is presented.

2.2 Software Placement as a CSP

Definition: A CSP [9], C, is a set of constraints, L, acting on a set of variables, $\Delta = \{A_1, A_2, ..., A_n\}$, each of which has a finite domain of possible values, D_i. A solution to L is an instantiation of all of the variables in Δ such that all of the constraints in C are satisfied.

We used the ChocoCP library [19] to solve CSP. Choco aims to minimize or maximize the value of a single variable, while respecting a CSP definition. To do this, it uses an exhaustive search based on a depth-first search. We used two CSPs to resolve the consolidation problem. The first CSP was used to determine the minimum number of VMs n_{new} needed to run all software; we call this the *MinVMToUse* problem. But n_{new} can be provided by several configurations (software mapping onto VMs). Therefore, the second CSP chooses the appropriate configuration and generates the reconfiguration plan to reach that

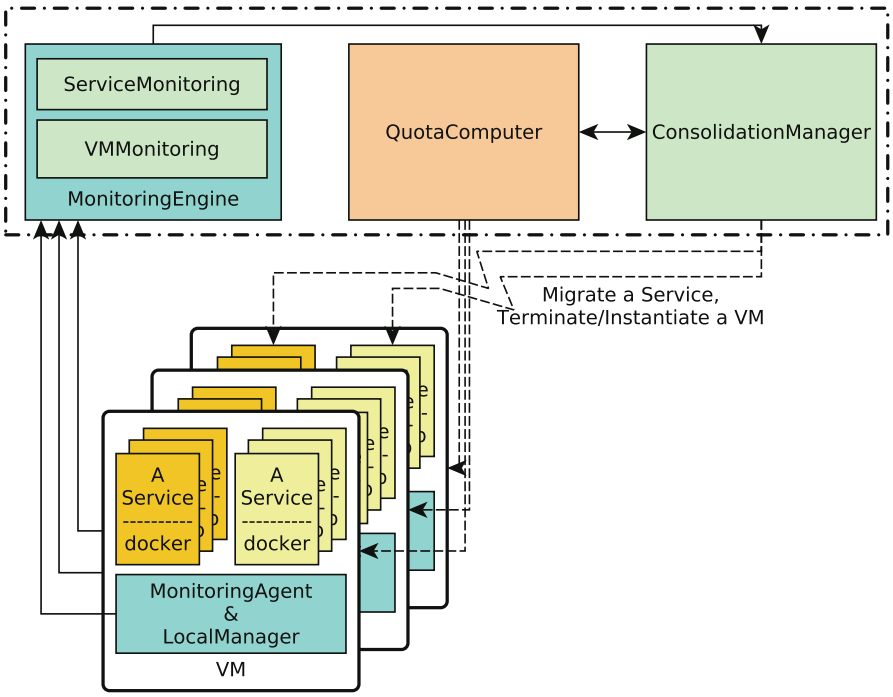

Fig. 1. Architecture of our software consolidation system

Algorithm 1. Software relocation

Begin

for each *software* s_i *of the SaaS/PaaS* **do**
  for each *resource type* r (u, m, *or* o) **do**
    if s_i^{rcur} *is insufficient and* $s_i^{rcur} < s_i^{rmax}$ **then**
      Increase the quota of s_i
    else
      if s_i^{rcur} *is excessive and* $s_i^{rcur} \neq s_i^{rmin}$ **then**
        if *The last quota decrease time is enough to avoid yo − yo effect* **then**
          Decrease the quota of s_i
        end if
      end if
    end if
  end for
  if *The actual VM of* s_i *does not have enough resources for − the new quota* **then**
    DestinationVM← The Best-Fit VM which can host s_i with its new quota
    if $DestinationVM == NULL$ **then**
      DestinationVM←Allocate a new VM
    else
      Disable any timer on DestinationVM
    end if
    Compute the docker container for s_i on DestinationVM
    Migrate s_i *to* $DestinationVM$
  else
    Update the docker container for s_i on its current VM
  end if
end for

End

configuration; this is called the *RightConfiguration* problem. We modeled these problems as a mixed-integer non-linear optimization problem. The inputs are a list of VMs with their total resources, a list of software (for each VM) with their current resource quota and status (service level they provide).

The *MinVMToUse* Problem. If no deployment request has been submitted, the number of VMs in use after application of the *ConsolidationManager* should decrease or remain the same. This should be done while avoiding resource over commitment. This is expressed in the following inequality:

$$\sum s_j^{u_{cur}} \leq vm_i^u \ \wedge \ \sum s_j^{m_{cur}} \leq vm_i^m \ \wedge \ \sum s_j^{o_{cur}} \leq vm_i^o, \tag{1}$$

$$where\ s_j^{vm} = i,\ \forall\ VM\ vm_i$$

We also allow the user to specify collocation requirements for each software. The following equation expresses that:

$$\mid s_i^{vm} - s_j^{vm} \mid + Col(s_i, s_j) \neq 0,\ \forall\ couple\ of\ software\ (s_i, s_j),\ i \neq j \tag{2}$$

$$where\ Col(s_i, s_j) = \begin{cases} 1\ if\ s_i\ and\ s_j\ are\ collocate-able \\ 0\ \text{otherwise} \end{cases}$$

The variable X minimizing the number of VMs is defined as follows:

$$X = \sum((len(vm_i) == 0)?0 : 1) \tag{3}$$

Speeding Up the Consolidation Process. We improved the consolidation process to reduce the solver execution time. First, we reduced the search domain for X by bounding it. In the best case, the minimum number of VMs is the sum of the resource quotas needed by all the software divided by the resource capacity of the biggest VM (we choose the most restrictive resource type). In the worst case, there will be no consolidation. This improvement is formalized as follows:

$$max(\left\lceil \frac{\sum s_i^{u_{cur}}}{max(vm_j^u)} \right\rceil, \left\lceil \frac{\sum s_i^{m_{cur}}}{max(vm_j^m)} \right\rceil, \left\lceil \frac{\sum s_i^{o_{cur}}}{max(vm_j^o)} \right\rceil) \leq X \leq n, \tag{4}$$

where n is the current number of VMs.

Second, some VMs or software may be equivalent in terms of resources or collocation constraints. If the resources offered by a VM, vm_i, are insufficient to host software s_j, then they are also insufficient to host any software s_k which has the same requirements. In addition, software s_j cannot be hosted by any other VM vm_k having the same characteristics as vm_i. With regard to the collocation constraint, if a VM, vm_i, runs software s_j which cannot be collocated with software s_k, then vm_i cannot host any software of the same type as s_k.

The *RightConfiguration* Problem. For correct configuration, the solver only considers configurations using the number of VMs determined by the first problem. The reconfiguration operation likely to affect the software SLA is live migration. The impact of this process could be a degradation of the service offered by the migrated software. Three factors affect live migration: network utilization, remaining computation power on both source and destination VM, and efficiency of the implementation of the live migration itself. Considering this, we call s_i^I the function calculating the impact of migrating software s_i for a given triplet of factors. Thus, if s_i^e represents the current service level provided by s_i before migration, then $s_i^e * s_i^I$ is the service level during migration. We define the cost of migrating a software s_i as $s_i^\Delta = s_i^e - s_i^e * s_i^I$. The correct configuration is the one minimizing K,

$$K = \sum s_i^\Delta, \ \forall \ software \ s_i \ to \ be \ migrated \tag{5}$$

while avoiding SLA violations:

$$s_i^e * s_i^I < s_i^T, \ \forall \ software \ s_i \ to \ be \ migrated \tag{6}$$

3 Use Cases

This work was conducted conjointly with two industrial groups: Scale Agent and Eolas. The former provides an implementation of the JMS specification, while the latter is a SaaS provider offering Internet services. We used our solution to manage a SaaS offering both a messaging service (such as IronMQ [3]) provided by Joram [4] software, and an Internet service based on a LAMP architecture.

Migrating a running software serving requests raises two main challenges that we had to address:

- (C1) Avoid loss of requests and state during migration.
- (C2) Make the migrated software available and accessible on the destination node after migration. This should be transparent for the clients.

Due to space limitation for this article, we present only the migration algorithm for the JMS server. Joram ensures that any message will reach its addressee within a configurable time window. We relied on this feature to complete the initial part of the first challenge (C1). For the second part of (C1), at runtime a Joram server keeps a persistence basis containing its entire state: processing messages, messages in transit, and processed messages. Therefore, a Joram server can be made available with the same state on the destination node by copying this basis from the source node to the destination node. With regard to (C2), in contrast to live migration of VMs, where the migrated VM keeps its IP address on the destination node, migrating software results in a new IP address (the IP address of the destination node). How can remote clients be transparently informed of this new address? In our system this is resolved by forcing clients to use the DNS name when dealing with the Joram servers. Thus, the accessibility

of the migrated server is provided by (1) dynamically updating the DNS server and (2) rebinding the JMS client to the DNS server. This is transparent to the client because the JMS client is implemented to automatically resolve new addresses after several attempts.

Immediate copy of the persistence basis can have an important impact on the service offered by the migrated Joram server when this file is very large. To avoid this problem, we have optimized the algorithm to transmit the log file block by block to the destination node. This optimization was inspired by the copy-on-write mechanism used for live VM migration. We customized the Joram implementation to dynamically integrate and evolve its state at runtime when receiving new persistence information. A timer, which is triggered at the beginning of the migration process, ends the copy to limit the duration of the whole process. This optimization is currently being integrated into the official implementation of Joram on the OW2 [4] open source platform.

4 Evaluations

We evaluated our solution to show the benefits of software consolidation on top of VM consolidation. These benefits are shown in terms of energy and cost savings. The efficiency and scalability of CSP-based consolidation methods were evaluated in [10,12].

4.1 Testbed Overview

The cloud testbed integrates both a private and a public platform. Our private cloud is a part of the Eolas data center. It is composed of 8 DELL PowerEdge R510 equipped with Xeon E5645 2.40 Ghz processors (one with a 12-core CPU, and the others with 8-core CPU), 32 Gb memory and 2 NICs at 1 Gbps. They are connected through a gigabyte network switch. The virtualized layer is provided by VMware VCenter 5.1.0 (ESXi 5) with the VM consolidation module DRS/DPM [1] enabled: a PM for the VCenter, a PM with an NFS server to host VM images and user sessions, and 5 PMs as ESXi to host VMs. The last PM hosts our system (including the DNS server) and the agents simulating the Joram and web server users. The cloud provides a single type of VM: 1 vcpu running at 2.4 GHz and 1 Gb memory. The public cloud used was Amazon EC2 in the M1, medium VM, configuration.

SPECjms2007 [6] was used to bench the Joram servers. It includes seven interactions. Thus 7 Joram servers (7 VMs) are needed to run it. The second use case was based on real traces of the Internet service (LAMP) offered by the Eolas SaaS.

4.2 Power Saving in the Private Cloud

We simultaneously ran 15 SPECJms2007 and 6 LAMP scenarios (up to 37 VMs) in two situations. In the first situation (noted WSC (With Software Consolidation)) we ran the experiment with both software and VM consolidation enabled,

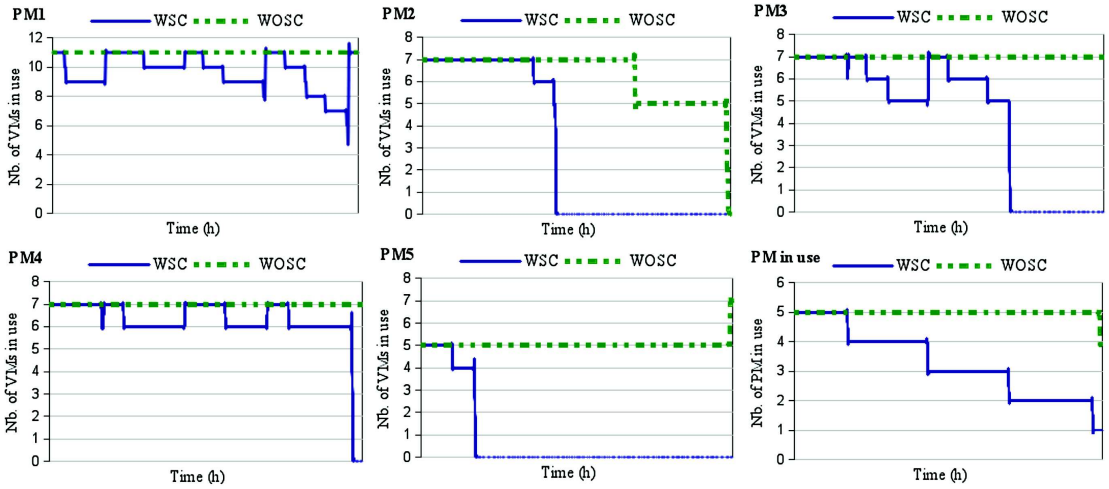

Fig. 2. Power saving in the private cloud: Utilization of PMs

while in the second situation (noted WOSC (WithOut Software Consolidation)) we disabled software consolidation (but maintained VM consolidation). The scenarios were configured to provide a varied workload over 30 h: a mix of constant, ascending and descending phases. Figure 2 presents (1) the occupancy (in terms of the number of VMs) of each PM in the private cloud, and (2) the number of PMs in use during the 25 h of observation. We see that the first situation results in 3 PMs (PM2, PM3 and PM5) being freed, while 1 PM (PM2) was freed in the second situation. Software consolidation accelerates VM consolidation. The bottom right curve in Fig. 2 shows that this improvement represents an approximately 40 % power saving with this particular workload.

4.3 Cost Saving in a Public Cloud

We repeated the previous experiments with VMs configured to run for an hour (before termination because they were empty) on Amazon EC2. We used M1, medium VMs instances, which are charged at $0.120 per VM per hour. Figure 3 presents the total number of VMs used over the 25 h of observation, and the total cost of the experiments. The number of VMs is seen to drastically decrease thanks to software consolidation, resulting in an approximate 40.5 % saving: from about $1300 (without software consolidation) to $800 (with software consolidation).

5 Related Work

Memory Footprint Improvements. Significant research has been devoted to improving workload consolidation in data centers. Some studies have investigated reducing the VM memory footprint to increase the VMs' consolidation, when a VM is dedicated to a single software. Among these, memory compression and memory over commitment ([8,15,22]) are very promising. In the same

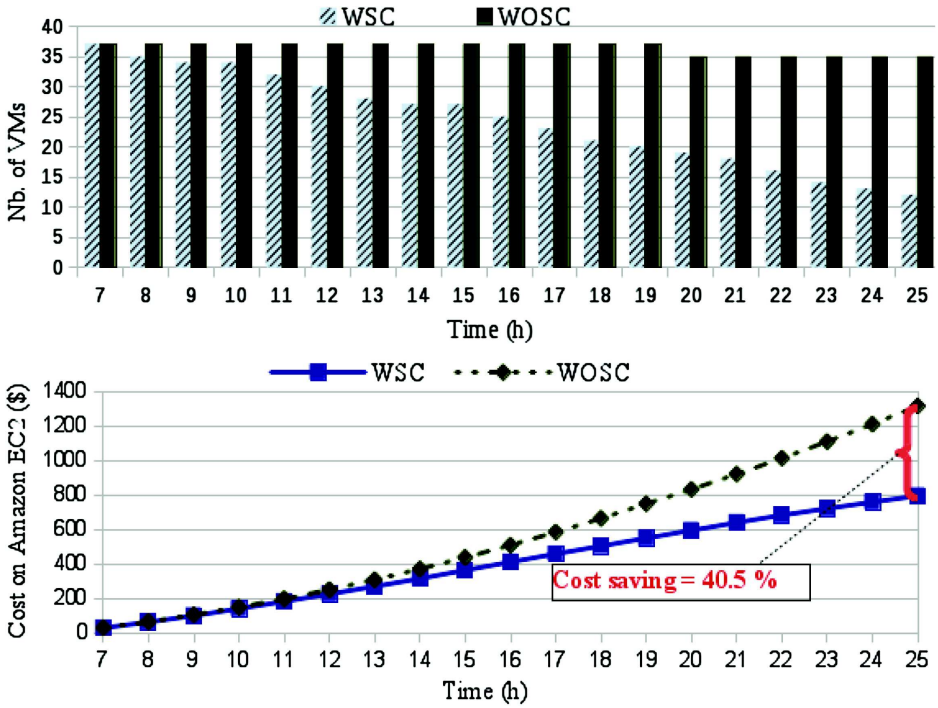

Fig. 3. Cost saving on Amazon EC2: (top) nb. of VMs per hour, (bottom) VMs charged

vein, [21] extends the VM ballooning technique to software to increase the density of software collocation on the same VM. [7] presented VSwapper, a guest-agnostic solution to reduce the effect of memory ballooning. Xen offers what it called "stub domain"[1]. This is a lightweight VM which requires very few memory (about 32 MB) for its execution. As our solution, all these works try do minimize the footprint of a VM in order to increase the number of VMs that can be collocated on top of the same physical machine. Therefore, they result to the same result as us in terms of energy saving. However, they do not minimize the total number of VMs as we do in order to reduce VMs charged for the clients when considering of a commercial cloud.

VM Consolidation Algorithms. In our previous work [23], we proposed a couple of this sort of VM relocation and collocation algorithms. [18] treats the VMs consolidation problem using a heuristic algorithm which minimizes the number of live migrations in the reconfiguration plan. An SLA-aware VMs consolidation system is presented in [11]. Like with our proposal, it formalizes the problem of minimizing the operating cost for a private cloud while also minimizing SLA violations for services offered by software. Our formalization can be extended by considering this work. [17] presents a VM consolidation strategy based on a predictive approach. Since the placement problem is NP-hard, it is not possible to develop a solution running within an acceptable time. [24] presents DejaVu, a consolidation system which takes into consideration the interference between

[1] http://wiki.xen.org/wiki/StubDom.

consolidated VMs. Based on hardware counters, it proposes a metric for characterizing workloads which are collocatable. In this paper, we do not focus on VM consolidation. We bring the same idea at software level (software within VMs). Therefore, any VM consolidation algorithm presented in this section can be applied to software consolidation. In this paper, we base on a solver to resolve the problem.

Software Consolidation. The main problem with previous solutions is that they are limited by the footprint of the VMs consolidated (they are all operating systems). Execution of a VM requires a set of minimum resources, even if the application it runs is idle. Thus, we propose a solution which dynamically packs software into VMs to effectively use the overall VM resources while respecting the individual requirements of the different software applications. With current knowledge, [10] is the only previous work that studies dynamic software consolidation on the same OS; however, it does not rely on VMs. [10] focuses on the MySQL database software and provides a live migration algorithm for that. This algorithm can be plugged into our framework. [10] (as well as Entropy [12]) describes a consolidation algorithm based on a CSP. Thus, no previous study has investigated software consolidation onto VMs. In this paper, we developed a working prototype and showed that it can achieve high VM utilization to provide cost and power-saving benefits.

6 Conclusion

In this paper we proposed a solution to consolidate software onto VMs to reduce power consumption in a private cloud and the number of VMs charged for in a public cloud. We focused on the algorithms for live migration and consolidation. Although the proposed solution can integrate other live software migration algorithms, we have provided a migration algorithm for JMS messaging. The consolidation algorithm is based on a Constraint Satisfaction Problem (CSP) approach. Evaluations with realistic benchmarks on a messaging and web applications SaaS cloud showed that our solution (1) reduces the power consumed by our industrial cloud partner by about 40 % when combined with VM consolidation, and (2) reduces the charge for VMs used on Amazon EC2 by about 40.5 %. Future work will include extended analysis of how best to coordinate software consolidation on VMs with VM consolidation on physical machines in order to further improve power gains.

Acknowledgment. This work was in part funded by IDS company, the French national program "Investissements d'Avenir IRT Nanoelec" ANR-10-AIRT-05 and Institut Universitaire de France.

References

1. Vmware distributed power management (DPM), Technical white paper (2010). http://www.vmware.com/files/pdf/DPM.pdf

2. Docker, June 2013. http://www.docker.com/
3. Ironmq: The message queue for the cloud, April 2013. http://www.iron.io/mq
4. Joram: Java (tm) open reliable asynchronous messaging, April 2013. http://joram.ow2.org/
5. Rightscale cloud management, April 2013. http://www.rightscale.com/
6. Specjms 2007, April 2013. http://www.spec.org/jms2007/
7. Amit, N., Tsafrir, D., Schuster, A.: Vswapper: a memory swapper for virtualized environments. In: ASPLOS, pp. 349–366 (2014)
8. Barker, S., Wood, T., Shenoy, P., Sitaraman, R.: An empirical study of memory sharing in virtual machines. In: USENIX ATC (2012)
9. Benhamou, F., Jussien, N., O'Sullivan, B. (eds.): Trends in Constraint Programming. ISTE, London (2007)
10. Curino, C., Jones, E.P.C., Madden, S., Balakrishnan, H.: Workload-aware database monitoring and consolidation. In: SIGMOD, pp. 313–324 (2011)
11. Goudarzi, H., Ghasemazar, M., Pedram, M.: Sla-based optimization of power and migration cost in cloud computing. In: CCGRID, pp. 172–179 (2012)
12. Hermenier, F., Lorca, X., Menaud, J.M., Muller, G., Lawall, J.: Entropy: a consolidation manager for clusters. In: VEE, pp. 41–50 (2009)
13. Karve, A., Kimbrel, T., Pacifici, G., Spreitzer, M., Steinder, M., Sviridenko, M., Tantawi, A.: Dynamic placement for clustered web applications. In: WWW, pp. 595–604 (2006)
14. Koomey, J.G.: Growth in Data Center Electricity Use 2005 to 2010. Analytics Press, Oakland (2011)
15. Liu, L., Chu, R., Zhu, Y., Zhang, P., Wang, L.: Dmss: a dynamic memory scheduling system in server consolidation environments. In: ISORC, pp. 70–75 (2011)
16. Lv, H., Dong, Y., Duan, J., Tian, K.: Virtualization challenges: a view from server consolidation perspective. In: VEE, pp. 15–26 (2012)
17. Mars, J., Tang, L., Hundt, R., Skadron, K., Soffa, M.L.: Bubble-up: increasing utilization in modern warehouse scale computers via sensible co-locations. In: MICRO, pp. 248–259 (2011)
18. Murtazaev, A., Oh, S.: Sercon: server consolidation algorithm using live migration of virtual machines for green computing. IETE TR **28**(3), 212–231 (2011)
19. Jussien, N., Rochart, G., Lorca, X.: The choco constraint programming solver. In: OSSICP, pp. 1–10 (2008)
20. Norris, C., Cohen, H.M., Cohen, B.: Leveraging ibm ex5 systems for breakthrough cost and density improvements in virtualized x86 environments. white paper, January 2011
21. Salomie, T.I., Alonso, G., Roscoe, T., Elphinstone, K.: Application level ballooning for efficient server consolidation. In: EuroSys, pp. 337–350 (2013)
22. Sharma, P., Kulkarni, P.: Singleton: system-wide page deduplication in virtual environments. In: HPDC, pp. 15–26 (2012)
23. Tchana, A., Tran, G.S., Broto, L., Palma, N.D.: Two levels autonomic resource management in virtualized iaas. FGCS **29**, 1319–1332 (2013)
24. Vasic, N., Novakovic, D., Miucin, S., Kostic, D., Bianchini, R.: Accelerating resource allocation in virtualized environments. In: ASPLOS (2012)